THREE-DIMENSIONAL HYBRID GRID GENERATION USING ADVANCING FRONT TECHNIQUES

John P. Steinbrenner and Ralph W. Noack
MDA Engineering, Inc.
Arlington, TX

ABSTRACT

A new 3-dimensional hybrid grid generation technique has been developed, based on ideas of advancing fronts for both structured and unstructured grids. In this approach, structured grids are first generated independently around individual components of the geometry. Fronts are initialized on these structured grids, and advanced outward so that new cells are extracted directly from the structured grids. Employing typical advancing front techniques, cells are rejected if they intersect the existing front or fail other criteria. When no more viable structured cells exist, further cells are advanced in an unstructured manner to close off the overall domain, resulting in a grid of "hybrid" form. There are two primary advantages to the hybrid formulation. First, generating blocks with limited regard to topology eliminates the bottleneck encountered when a multiple block system is used to fully encapsulate a domain. Individual blocks may be generated free of external constraints, which will significantly reduce the generation time. Secondly, grid points near the body (presumably with high aspect ratio) will still maintain a structured (non triangular or tetrahedral) character, thereby maximizing grid quality and solution accuracy near the surface.

INTRODUCTION

Grid generation has turned out to be a significant aspect of the computational simulation of field problems. A cursory literature search would produce a variety of fundamental grid types in mainstream use, which is an indicator that no one method offers clearly superior features. The traditional method of grid generation for computational fluid dynamics, structured grid generation, remains popular today because it provides substantial solution accuracy, particularly in viscous regions of the flowfield. A generalization of the structured method, the multiple block method, allows virtually any geometry to be modeled with a series of abutting grid blocks. This geometrical flexibility comes at a price, however, as the person-hours needed for generation can become exhorbitantly high. This one fact was probably the major reason for the ground swell of unstructured grid generation CFD applications in the last eight or so years. Unstructured grids do indeed ameliorate the generation-time bottleneck, but there has been a conspicuous delay in application of unstructured grids to viscous regions, where the severe length-scale disparities of the problem necesitates cells of very large aspect ratio. Even now that viscous applications are appearing in the literature, the memory and computational overhead associated with an unstructured grid makes it difficult to generate grids of sufficient resolution. In 3D, for example, it takes 5 or 6 tetrahedral cells near the body to replace a single hexahedral cell, and even then the tetrahedra will usually exhibit high aspect ratios.

Much of the grid generation research of the past few years has been aimed at alternate methods that allow grids to be generated in reasonable times without compromising resolution and hence, solution accuracy. Perhaps the most mature of these methods is the overset, or Chimera approach. Here, structured grids are generated independently about different components of the geometry in an overlapping manner. Overlap regions are then determined either from user-input, or lately from automated means [1], and solutions are obtained on the composite grid. The difficulty of this method is in the conservative interpolation of the flow solution from one component grid to another. Complex methodologies have been developed for this step. Another emerging grid technology is the cartesian approach, whereby the flow domain is automatically

divided into quadrilaterals and parallelepipeds at locations aligned with the physical coordinates. Here, local refinement is often used for enhanced accuracy. The current problem with the cartesian method is the representation of the geometry surfaces, which are usually not aligned with the physical axes. The typical remedy is to modify the cell locally near the surface to maintain a body-conforming fit.

This last fix introduces yet another emerging grid technology, that of the grid of mixed structure, or the "hybrid" grid. The most popular hybrid grid application is to combine regions of structured grids with regions of unstructured grids in such a way to preserve the salient features of each method. For example, by using structured grids near the geometry surface, and unstructured grids everywhere else, the inner grid could be generated quickly, and the outer grid could be generated automatically, thus eliminating the difficulty of constructing a multiple block topology that links several blocks together to represent the entire domain. The structured character of cells near the body would then lead to improved solution accuracy, while representing the domain locally with fewer grid cells. Early hybrid grid examples were applied to geometries with clearly defined near and farfield regions [2], with the non-overlapping structured grids generated first, and the remaining regions filled in later with unstructured cells. More recent examples, applied to extremely complex geometries, still require user-defined boundaries between regions of structured and unstructured cells [3]. In the general case, however, the line of demarcation between structured and unstructured regions is not clearly defined, and in fact, may be impossible for the user to specify. The most intuitive solution is to allow the near-geometry region to be generated automatically, until a specified distance is reached or until intersections are found. Very impressive grids have been developed recently using this approach [4], [5], but as the generation of surface and near-surface grids is the toughest aspect in *all* types of grid generation, it may be some time before grids of this type may be generated automatically for general geometries and flow conditions.

Another means of generating hybrid grids is to generate a series of structured, overlapping grids around different components of the geometry, similar to the chimera approach. Here, however, grid cells in the overlapping regions of the grid will be removed, and will be replaced with unstructured cells. The resulting hybrid grid will then fully encapsulate the flow domain with no regions of overlap. Determination of regions of overlap becomes the central issue in this approach. One effort employed well-established chimera tools for blanking out the regions of overlap between grids [6]. The chimera tools used, however, required the grids to be sorted into major and minor components, which forced users to be familiar with the manner in which the component grids overlapped. For obvious reasons, requisite user-comprehension of block overlap will impede the generation of hybrid grids around general configurations, and so an automatic means of overlap detection is desirable.

The method proposed in this paper for hybrid grid generation manages grid overlap automatically through the use of an advancing front algorithm. Specifically, fronts are formed on surfaces of the structured blocks, and are then advanced by covering structured cells that do not intersect the front.[†] The end result is a collection of structured cells that cover the flow domain in all regions except for the gaps where opposing fronts nearly converge on each other. The gaps are in turn discretized with unstructured cells by applying another advancing front solution, this time using unstructured cells.

The remainder of this paper explains the details of the advancing front hybrid approach, and provides both 2 and 3 dimensional applications.

[†]The basic idea of using advancing front techniques on a series of structured grids is also presented in Reference [7]. In that chimera grid application, the advancing front determined regions of the grid to be blanked out in the composite grid, and limited the overlap (a chimera requirement) rather than eliminated it.

BASIC ADVANCING FRONT METHOD

Generation of hybrid grids in this study is based on advancing front methods traditionally used for unstructured grid generation. Hybrid generation consists of two successive applications of a general advancing front algorithm- the first advancing along a predefined set of overlapping structured grid data, resulting in a non-overlapping set of structured cells, and the second advancing along previously unfilled regions of the domain using unstructured methods. Despite the inherent differences in these two applications, each employs the same basic steps:

1. Form an initial front as a closed-loop of faces.
2. Identify a face (*pface*) on the front from which a cell is to be constructed.
3. Form a candidate cell, *cell1*, using the *pface*.
4. Test the candidate cell for validity.
5. If the candidate cell is valid, add the cell to the grid, and modify the front so that the new cell lies behind it. This requires removing faces that are covered by *cell1*, and adding the remaining faces of *cell1* to the front.
6. If the candidate *cell1* is *not* valid, find another candidate cell and go to step 3.
7. Repeat the process until there are no more faces on the front, or until there are no more suitable *pface*'s.

The specifics of the advancing front method for both structured and unstructured grid regions will be detailed in subsequent sections.

COMMON DATA STRUCTURES AND ALGORITHMS

There are so many fundamental similarities between the two advancing front applications that the current method attempts to utilize common data structures and routines in generating the structured and unstructured grid regions.

Data Structure

There are three basic geometric entities, each with an associated data structure. The smallest discretization of the overall problem domain is known as a *cell*, which represents an area in 2D and a volume in 3D. Cells are bounded by a number of *faces*, which represents a length in 2D and an area in 3D. The edges of the faces are then represented by either a single *node* in 2D, or a line bound by two nodes in 3D. Nodes are the basic geometric entity, and their data structure contains an (x, y, z) coordinate triple indicating location in physical space. A face data structure, on the other hand, contains a list of forming nodes (2 in 2D, 3 in 3D) as well as a pointer to the two cells formed on each side of the face. Finally, a cell data structure is composed of a list of faces and nodes. The number of faces per cell depends on the dimension of the problem and on whether the cell is structured or unstructured. Triangular (2D) and tetrahedral (3D) cells are used for unstructured cells, and quadrilateral (2D) and hexahedral(3D) cells are used for structured cells. Figure 1 lists the number of faces required to form the different cells used in the present work. Note that there are twelve faces in the hexahedron cell because each hexahedral side is broken into two triangular faces in order to utilize the same face intersection routines (described below) as the tetrahedral mesh generator.

The final data structure to discuss is the *front*. The overall front is a list of *front* entities, each of which contains a face and the associated connection data that specifies the face's neighbors at each edge. Additional geometric data is stored to facilitate the various searches required by the method. A bounding sphere radius and center is stored for each cell and face on the front to assist in face intersection and proximity tests

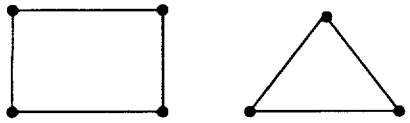
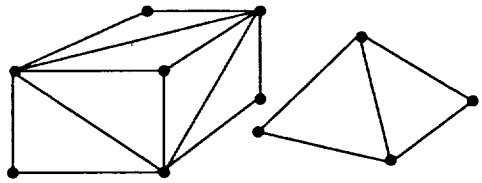
2D		3D	
			
quadrilateral	triangle	hexahedron	tetrahedron
2 nodes/face	2 nodes/face	3 nodes/face	3 nodes/face
4 faces/cell	3 faces/cell	12 faces/cell	4 faces/cell
faces are lines	faces are lines	faces are triangles	faces are triangles

Figure 1: Cell structures for 2D and 3D.

described later. A front face also stores the normal to the face, area of the face, target cell size evaluated at the face centroid, and the angle between the face and its neighbors. A *state* indicator is also stored at each face on the front and reflects how close the face and one or more of its neighbors are to forming a complete cell. State values will be described in depth later in the structured and unstructured advancing front details.

Quadtree/Octree Data Storage

The advancing front method requires a significant number of global data searches. Two examples include searching for a list of all nodes on the front that might be suitable for forming a cell, and searching for the list of candidate faces that might intersect a candidate cell. Typically several searches of this type are needed for every iteration of the advancing front. Clearly then, global linear searches for all of these possibilities is prohibitively expensive. The present approach to speed these searches is to use octree (3D) and quadtree (2D) data structures [8], so that linear searches are localized to a relatively small vicinity. Currently the nodes on the front and a list of faces containing the node are stored in one octree/quadtree structure, and the front faces themselves are stored in another, with the face centroid used as the storage location. The bounding box for the quad/hex containing the list of faces is expanded to contain the bounding sphere for all the faces contained in the quad/hex. Thus when searching for the list of faces whose bounding spheres intersect the bounding sphere of a cell, the cell bounding sphere is tested against the bounding box of a quad/hex in the tree. Those bounding boxes that do not overlap do not need to be tested further.

Intersection Tests

The validity of a candidate cell, *cell1*, formed by advancing the front depends upon accurate testing for intersections between *cell1* and the faces on the front. The first step is to determine a set of faces on the front that are likely to intersect the faces of *cell1*. The bounding spheres for the candidate cell and the faces on the front are utilized in the search for a set of faces that that could possibly intersect the candidate cell. Since the cell and a face are completely contained within the bounding sphere, if the spheres do not intersect, then neither will the cell and faces. Hence rigorous intersection tests need only be applied to the list of all faces whose bounding spheres intersect the bounding sphere of the candidate cell.

The intersection test between a cell and a face begins by checking if the edges of the cell intersect the face and conversely, checking if the edges of the face intersect the faces of the cell. Additional tests are performed to reject the candidate *cell1* if the edges of the cell lie within the face or if the edges of a face lie within the

faces of the cell. Finally, *cell1* is rejected if any nodes on the front lie within the volume of the cell.

HYBRID FLOW SOLVER

An extensive development effort parallel to this one is currently underway for a 2 and 3 dimensional Euler and Navier-Stokes solver for use with the hybrid grids generated herein. The flow solver adopts a cell-centered finite-volume formulation and an unstructured face-based grid connectivity that allows each cell to have an arbitrary number of bounding faces and allows each face to have an arbitrary number of forming nodes on its perimeter. The system of equations is solved with an implicit point Gauss-Seidel relaxation scheme. An upwind difference flux-vector splitting scheme is used to define the numerical flux at each cell interface. A second-order accurate upwind-biased extrapolation of the primitive flow variables defines the left and right states at the cell face using a cell-averaged gradient computed with a Green-Gauss integration of the solution reconstructed at the forming nodes. This flow solver is described in detail in [12].

GENERATION OF STRUCTURED MULTIPLE BLOCK GRIDS

A major advantage hybrid grids afford the user over multiple block structured grids is the ability to generate block grids independently for different components of the geometry, thereby eliminating the unwieldy topological constraint of point-to-point matching at block boundaries. In this hybrid approach, these component block grids are assembled together, forming regions of single grid representation, regions of grid overlap, and regions of no grid representation. While these last two types would preclude the multiple block approach, both are handled with this hybrid algorithm, which automatically removes overlapping regions, and which fills gaps between grids with unstructured cells.

Still, the structured grid remains the starting point for the hybrid process, and serves as the "skeleton" for the final grid. In this method, structured grids are generated in multiple block clusters, with each cluster containing a series of blocks that abut one another exactly over a portion of the blocks' surfaces. The limiting case of single block clusters is permissible, so single blocks may be considered a subset of multiple blocks for this scheme. Fortunately, a multiple block grid cluster is sometimes as easy to generate as an individual block, and the quality of grid found in a multiple block cluster is worth preserving in the hybrid grid. For example, in Figure 2, the two abutting blocks (D and E) around the main airfoil require no more work to construct (perhaps less) than would several overlapping blocks around the same airfoil. The proper strategy, then, is to generate multiple block grids only in regions where the topology does not markedly impact generation time.

Numerous tools are readily available for the construction of multiple block grids. All grids in this work were generated with Gridgen [9], [10], a widely-used general-purpose software package designed for use with a suite of public-domain flow solvers. Gridgen automatically detects regions of block-to-block connections, and also provides a graphic user-interface for the establishment of flow solver boundary conditions on block surfaces. Further, it exports ASCII grid data and connection data that is directly readable by the selected flow solver. This feature was exploited in the development of the hybrid solver, which adopted Gridgen's generic file formats for both grid point and connection data. This allows the user to set up the majority of the hybrid grid generator inputs directly in Gridgen. Keep in mind, however, that any multiple block grid generator would work as well, once the exported data were translated to the proper format.

HYBRID GRID GENERATION

The generation of the hybrid grids from the predefined set of structured grid points follows a linear path that begins with converting the structured grids into hybrid components, and continues with parsing these points into non-intersecting regions, forming an initial front, advancing the front along structured cells,

removing unused cells, massaging the front and then advancing it further using unstructured cells.

Importing Grids into Hybrid Structure

The hybrid algorithm commences with a conversion of the raw grid data generated in multiple block form into the node-face-cell paradigm required by the hybrid solver. Conversion to the hybrid format follows the multi-step path described below.

For the trivial single block case, the remainder of this section is unnecessarily complex, since connections will not exist, and the block's nodes, faces and cells could be generated automatically on the basis of their implied connectivity. In the more general (and more interesting) multiple block case, however, surfaces from different blocks will abut one another, with any number of block edges and corners sharing nodal values. In these cases it will not suffice to treat blocks independently, lest multiply defined nodes and faces be formed on the block's abutting regions.

Construction of the hybrid entities begins with node formation, and continues with face and then cell formation for *all* blocks in the system. No consideration is given to component overlap or intersections during this phase. Rather, the entire structured grid is fit into a hybrid structure that will later serve as the template on which the advancing front algorithm will be applied. This later phase will determine overlap and will act accordingly.

Connection Arrays.- The interblock connection file associated with the multiple block grid is read into a series of arrays as a preliminary step. These arrays specify flow boundary conditions or connections with other blocks for each bounding surface of the block. Connections and BC's may be applied at the partial surface level, meaning that more than one BC or connection may be applied to a particular block boundary. Next, blocks in the system are divided into groups based on topological linkages defined in the connection arrays. Two blocks of the same group indicate that it is possible to get from one block to the other while crossing only at connections specified in the arrays. Figure 2 illustrates the block group concept for a 6 block, 3 group example. Although block groups make no guarantee about the relative geometric positioning of the blocks, as a matter of practice it is assumed that blocks of the same group contain no region of overlap. This is not unreasonable in consideration of the fact that block groups will generally correspond to components of the geometry, and will likely be generated independently of other groups. This assumption of no overlap within block groups will be exploited later to reduce the number of intersection tests during front advancement.

Formation of nodes.- Nodes are the most difficult hybrid component to construct from a set of multiple-block structured grid data because there exists no one-to-one correspondence between grid points and nodes. In Figure 3, for example, grid points from three individual blocks are all represented by the same hybrid node value. In contrast, at most two structured grid surfaces will be represented by a hybrid face, and a hybrid cell will always represent a unique region of a structured grid.

To facilitate the placement of nodes into face and cell components, a temporary rectangular array of Null-initialized node pointers $node\_ptr[i, j, k, n]$ is formed for each block in the system, sized to the computational dimensions of the block. The i, j, k, n indices correspond directly to the structured grid point with like-indices, with n referring to block number. The $node\_ptr$ array is filled easily for indices on the interior of the block, since interior points will correspond to unique nodes. For these indices, a new node is allocated, and its address is assigned to $node\_ptr[i, j, k, n]$. For indices corresponding to block extremities, however, more care is taken. If the value in $node\_ptr$ is Null, a list of *all* grid points using the node is formed, a new node is allocated, and its address is assigned to $node\_ptr$ at each grid point index in the grid point list. On the

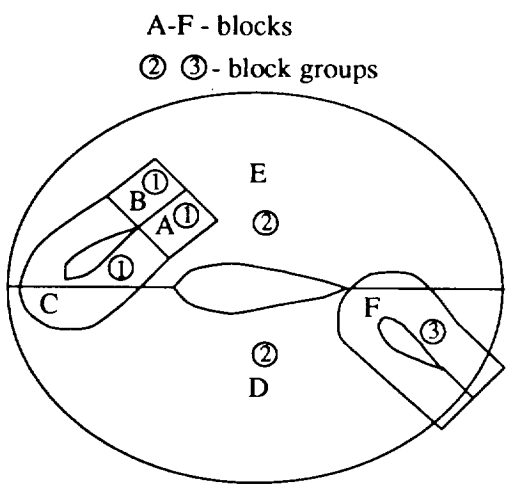


Figure 2: 6 blocks forming 3 block groups.

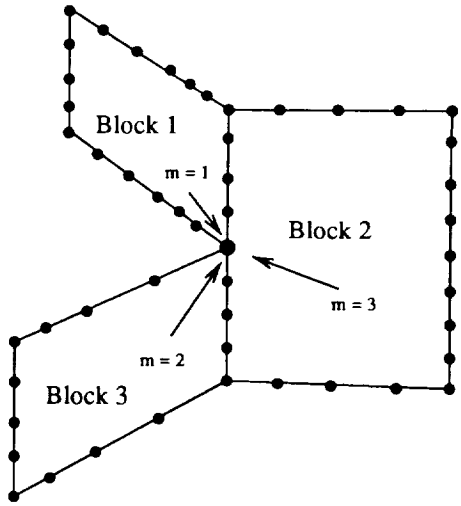


Figure 3: Nodes common to blocks.

other hand, if the value in *node_ptr* is not Null, a node is already assigned to the grid point, and the next index is considered.

The list of block grid points sharing a node is determined via an iterative algorithm that traverses the outer surfaces of the structured blocks, searching for minimum surface elements that use the node. A minimum surface element is defined here as a 2 by 2 patch (2 by 1 in 2D) of adjacent grid points on the block boundary. Define also a *POF* (point on face) as an artificial entity containing i, j, k, n ranges of a minimum surface element, and indices to one of the element's corners.

Now, for each grid point i, j, k, n index on the outer surfaces of the structured blocks, a *POF*, labeled A, is formed at the index. A is pushed onto a stack, and the iterative loop begins.

1. Look through the stack for the first unused *POF*, called P. Mark P as used. If there are no unused *POF*'s, exit.
2. Using the connection arrays, determine if P abuts to another block. Call this "image" *POF* I. If it exists, check I for uniqueness with all other *POF*'s on the stack. If unique, add it to the stack, and mark it used.
3. There will be up to 3 other *POF*'s on the block boundary that will be adjacent to P and will share the same grid point index (up to 1 *POF* in 2D). Add each one of these "neighbor" *POF*'s to the stack if they are unique, and mark them as unused.
4. Similarly, look for up to 3 other *POF*'s on the block boundary that are adjacent to I and share the same grid point index. Add each one of these "neighbor" *POF*'s to the stack if they are unique, and mark then as unused.

When this algorithm is exited, *all POF*'s corresponding to a particular hybrid node will be determined. The final step is to scan the list of *POF*'s one more time, removing any *POF*'s that correspond to the same i, j, k, n grid point index. What is left is a list of all block grid points that wil share the same nodal value. This algorithm is illustrated in Figure 4, which tracks the 4 iteration history of determining the grid points associated with a particular node.

Get the first unused POF on the stack, called P. Mark as USED.	A	D	E	F	-
Using connection info, find the image of P, called I. Mark as USED, add to stack if unique.	B	-	C	-	
Find neighbor(s) of P sharing node i on the boundary. Mark as UNUSED, add to stack if unique.	D	A	B	C	
Find neighbor(s) of I using node i. Mark as UNUSED, add to stack if unique.	E	-	F	-	

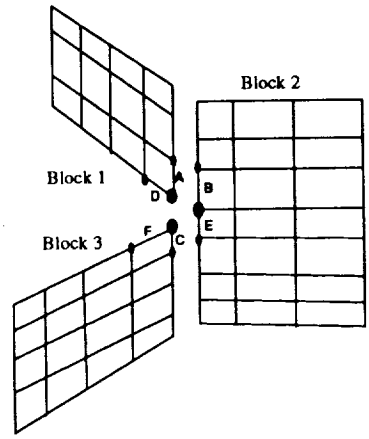


Figure 4: Algorithm for determination of common nodes.

Formation of cells.- Hybrid cell formation from structured grid data proceeds by allocating a temporary rectangular array of Null-initialized cell pointers $cell\_ptr[i,j,k,n]$ for each block in the system, sized to the cell dimensions (grid point dimension minus one) of the block. Each array is then filled by allocating a new cell and assigning its address to $cell\_ptr$. Nodes are then assigned to these cells by using the $node\_ptr$ data at the same i,j,k,n index in addition to its neighbors $((((node\_ptr[i+a,j+b,k+c,n],a=0,1),b=0,1),c=0,1))$

Formation of faces.- Face formation also proceeds by traversing the $cell\_ptr$ arrays at all possible cell indices i,j,k,n. The cell data structure contains pointers to all 12 (4 in 2D) of its constituent faces. At a given i,j,k,n index, heretofore unassigned faces are allocated and assigned to the cell. Faces interior to blocks are also assigned to the cell adjacent in the $cell\_ptr$ arrays. Faces on the boundary of blocks are checked for abutment with other blocks via the connection arrays. If such a connection exists, the newly allocated face is assigned to the abutting cell.

Treatment of singularities.- In order to fully represent a component of a geometry with structured multiple blocks, it is often convenient to introduce regions of singularities, or poles. Grid points lying along a 3D axis of symmetry, for example, will locally collapse to a region of zero area in the circumferential direction. Fortunately, these regions are easily identified and may be set with a Pole-type boundary condition in the connection arrays.

During node formation above, a node is allocated for each point along the singularity axis within the specified Pole BC regions and is assigned to $node\_ptr$. This $node\_ptr$ value is then applied to all remaining grid points in the direction of the singularity, thereby preventing the duplication of nodes. During face formation, then, if the candidate face accesses a given node more than once, the face will have zero area, and will be discarded and removed from the cell(s). This in turn modifies the local form of the cell, which could be reduced from 12 to as few as 4 faces in 3D, or from 4 to 3 faces in 2D.

Identification of Non-Intersecting Subblocks

All structured grid points have now been loaded into the hybrid entities, namely the nodes, faces and cells. In the next section, some of these faces will be grouped and linked to form initial fronts for the advancing front algorithm. As will be shown, the front is advanced by swallowing one cell at a time, employing a series of intersection tests to ensure that the new front does not intersect the previous front. This is a relatively expensive test that is applied to the complete front once for each non-intersecting cell, and one that is clearly

not needed if it is known *a priori* that the cell in question does not violate intersection tests.

In light of this, a simple preprocessing step based on rectangular-extent or bounding-box testing [11], is employed to identify regions of the grids quickly that do not overlap or intersect with any others. Figure 5 depicts a collection of non-intersecting cells determined purely from these tests, described below.

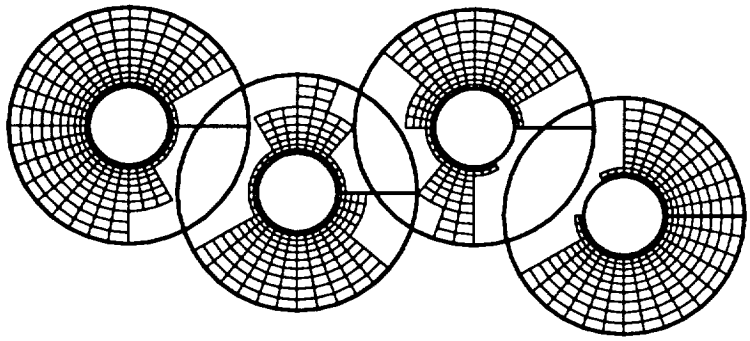


Figure 5: Non-intersecting subblocks.

Formation of subblocks.- The subblock entity is a data structure containing the computational and physical extents of part of a block. Initially, one subblock is formed for each block in the system, extending over the entire block, from 1 to $(imax, jmax, kmax)$. Each grid point in the subblock is then examined to formulate the range of x, y and z values for the subblock (the bounding box).

Splitting subblocks.- Next, a number of independent subblocks is constructed through successive subblock splitting. A subblock is deemed independent if its bounding box does not intersect the bounding box of any other subblock with a differing block group number. The test for subblock intersection is trivially performed due to the cartesian shape of the bounding boxes.

When a subblock is found to intersect another, it is split along its longest computational dimension, and new bounding boxes are computed for each subblock. The subblock is split as long as it does not violate a user-prescribed (usu. 3) minimum size. If an intersection is not detected, the subblock is independent and flagged as such. The procedure ends when there are no more dependent subblocks left to split.

Concatenating subblocks.- The previous step generally produces a large number of small independent subblocks, and so the subblocks are concatenated in this step to reduce their number. Concatenation takes place by checking the computational range of each edge of each subblock for an identical range elsewhere in the list. A match indicates two fully abutting subblocks, which are combined into a larger subblock. The procedure continues until there are no two remaining subblock edges that represent the same surface.

Remove overlapping subblocks.- Finally, all subblocks flagged as dependent are removed, resulting in a list of block ranges on which there is known to be *no* overlap or intersection. All cells within these independent subblocks are marked for easy disposition later.

Formation of Initial Front

The collection of newly formed cells, faces and nodes represents a superset of the structured entities that will comprise the hybrid grid. This superset will be pared down to a non-intersecting set of entities by advancing a series of initial fronts one cell at a time, rejecting cells along the way that violate overlap and

proximity criteria. There are two types of initial fronts- those specified by the user and those representing bounding surfaces of non-intersecting subblocks formed earlier. Both types of fronts must form closed surfaces, so that it will remain simply connected as it covers cells. Such a requirement will eventually allow some of the front to disappear, signaling the end of the algorithm.

The faces in the system which will comprise the initial fronts are specified in a list of structured surface regions. User-specified front regions are passed in through the connection arrays, and generally will correspond either to the geometry surfaces or to the farfield extents of the overall domain. A total of six (4 in 2D) regions (corresponding to the block boundaries) are also added to the list for each remaining subblock. Subblock boundaries are chosen as initial fronts because they represent boundaries of regions of no overlap, and subsequent front advancement will move away from the boundaries, thereby eliminating a large number of unnecessary intersection tests.

Initial fronts are formed in two sweeps. In the first, each surface region, which represents a computationally rectangular surface on the structured blocks, is swept. Each face in the region will be loaded into the front, and will be connected to the faces with adjacent i, j, k, n index, via the *cell_ptr* arrays. When the region is loaded, only the region's bounding edges will not be connected to other faces.

Special care is needed for pieces of front regions that are doubly-defined. This situation occurs at common surfaces of subblocks, and at subblock surfaces also user-specified as initial fronts. In all cases, a doubly-defined front face should be removed from the front, and so if a face to be added is already on the front, it is instead removed, and its neighbor's connections are broken.

In the second sweep, the edges of front regions and doubly-defined regions are stitched together to ensure a series of simply-connected initial fronts. This is accomplished by interrogating each edge of each front face. If the neighbor to the face at the edge in question is not defined, a search is made to find all faces on the front using the edge. If two edges are returned, their corresponding faces are linked together, and the next edge is found. The case when more than two edges are returned indicates an improperly formed initial front, at which time the algorithm is aborted.

Temporary arrays *cell_ptr* and *node_ptr* may be discarded after front initialization. From this point on, all connectivity between cells, faces and nodes is transmitted from within their respective data structures.

Advancing Front Along Structured Cells

Except for the user-specified initial front selection. all of the steps described above proceed in an automatic fashion. When these steps have completed, the advancing front algorithm is started along the structured cells. Front advancement follows an iterative three step procedure until the front has disappeared or until it may be advanced no more. In the first step, a suitable face on the front is selected for front advancement. Secondly, the cell in front of this face (the cell to be covered) is tested for geometric violation with other faces on the front. Finally, if it passes the test, the front is modified to include the cell, and the front face connections are modified to maintain a proper front.

Candidate face selection.- The first step in the iterative structured front advancement procedure is to select a face, called *pface*, from which the front is to advance. The importance of *pface* selection should not be underestimated, as differing selection criteria may produce hybrid grids of widely varying character. This is because front advancement at a given location precludes advancement of an opposing front that is vying to cover the same physical space.

Penalty function: Since the proper selection of *pface* is influenced by a host of conflicting geometrical

and topological factors, it is logical to assemble these factors into a mathematical "penalty" function, so that *pface* may be selected as the face of minimal functional value. In this work the linear expression below is used, where n refers to the number of component functions, c_i is a user-set non-negative influence coefficient, and f_i is the normalized $i'th$ component function.

$$F_{penalty} = \sum_{i=1}^{n} c_i f_i$$

There are currently two component functions used in the overall penalty function. The first, f_{size}, measures the relative size of the face. Specifically,

$$f_{size} = \frac{log(S/S_{min})}{log(S_{max}/S_{min})}$$

where S is the area (length in 2D) of the face, and S_{min} and S_{max} are the minimum and maximum areas among all faces on the front. Notice that f_{size} is bound by 0 and 1. This function produces a smaller penalty for faces of smaller size, but the logarithmic form reduces order of magnitude disparities in face size to a linear scale, preventing over-penalization of the mid-size faces.

The second component function, f_{state}, measures the local "raggedness" of the front. Here, a "state" variable is formed at each face in the cell, set equal to the number of front neighbors that touch the same cell outside the front. Front faces with larger values of the state variable result in a local "stair-step" construction of the front, as depicted in Figure 6. This local phenomenon is to be avoided, if possible, because it reduces the advantage of local structure in the hybrid grid, and it creates a more challenging starting front for the unstructured grid generator described later. Therefore, the goal of f_{state} is to minimize the penalty at faces that tend to fill in the ragged edges of the front. This suggests the following form:

$$f_{state} = 1 - \frac{T}{T_{max}},$$

where T is the state value, and T_{max} is the maximum state value found on the front.

In practice, f_{size} seems to be the more important function, and is typically used with coefficients of $c_{size} = 1$ and $c_{state} = 0.25$. This combination insures that the *pface* will be chosen on the basis of state variable when the face sizes are nearly the same.

By no means are the two functions above sufficient for front face selection in all cases; this is in fact one of the lesser mature aspects of the advancing front hybrid approach. Other penalty function components for future consideration include the proximity of the front face to opposing fronts and the variance of the forming cell size from a prescribed target volume.

Cycling: During the early stages of front advancement, it is desirable to have the front grow in a fairly uniform manner, so that viscous layers of the structured grid may be covered by the front before overlap in the grids becomes a factor. This is accomplished by associating a *cycle* variable to all front faces, roughly equivalent to the number of layers from the initial front that the current face lies. A global *cycle* value is also formed, equivalent to the layer that is currently being filled. As the front advances, only front faces with the current value of *cycle* become candidate *pface*'s, and the winning *pface* is selected from the candidates using the penalty function approach described above. After *pface* is covered by the front, any new front faces are given an incremented *cycle* value so that they will not become candidate *pface*'s until the next layer (global *cycle* is incremented). When there are no more front faces with the global *cycle* value, the *cycle* value is

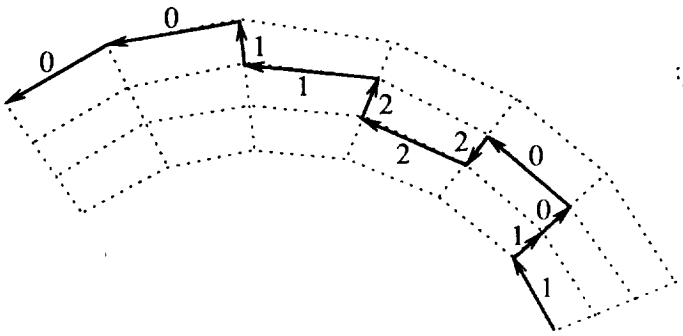


Figure 6: State values along a typical front.

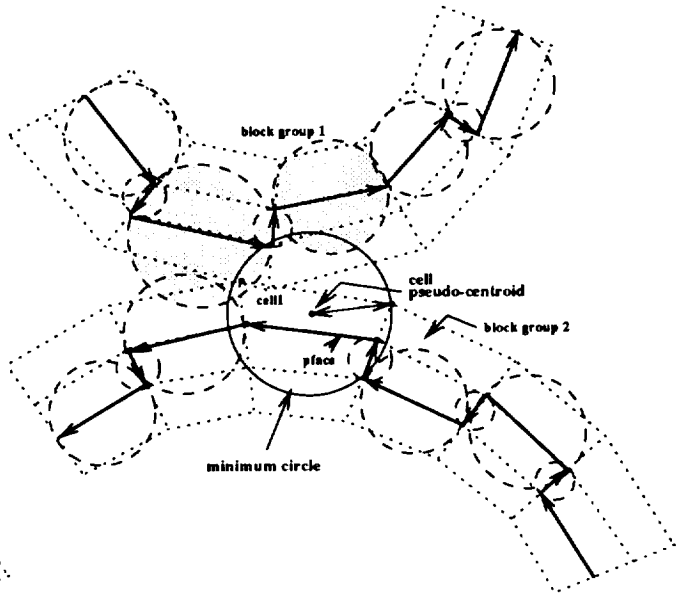


Figure 7: *cell1* is rejected due to proximity to shaded circles.

incremented, and the procedure is repeated. When the number of user-specified cycles is met, all front faces become candidate *pface*'s henceforth.

A typical value for *cycle* is 3, which guarantees 3 layers of grid points around initial fronts, barring front violations, described next.

Violation checks with front.- Before the structured front is advanced at a given *pface*, the faces of *cell1*, the cell outside the front at *pface*, are checked for two types of geometric violations, described below, with the rest of the front. If a violation is detected, the front is *not* advanced at *pface*, and *pface* is flagged so that it will not be a candidate face in subsequent iterations.

Strict intersection: All faces on *cell1* not already on the front are checked for intersections with all faces on the front as described earlier. If an intersection is found, *cell1* clearly *may not* be covered by the front.

Proximity: Experience has indicated that it is also necessary to keep fronts from getting "too close" to each other. This is driven by the fact that the completed structured front will serve as the starting front for the unstructured generator, and a finite distance between fronts is needed to provide adequate space for the unstructured front to form reasonable cells.

Definition of the term "too close" is understandably nebulous. The approach herein is to calculate the pseudo-centroid of the *cell1* by averaging the node values, and then to form a sphere (circle in 2D) of minimum radius that surrounds all nodes. Next, for each face on the front, the sphere (circle in 2D) of minimum radius that contains all nodes on the face is calculated. The front face minimum spheres are then scanned for intersections with the minimum *cell1* sphere, ignoring front faces belonging to the same block group as the original *pface*. Front faces of the same block group are ignored because front faces adjacent to *pface* would almost always be intersecting, and it is implicitly assumed that blocks of a common block group contain no overlap.

This type of "proximity" check is schematically illustrated in Figure 7, and its effect is shown in the

example of Figure 8.

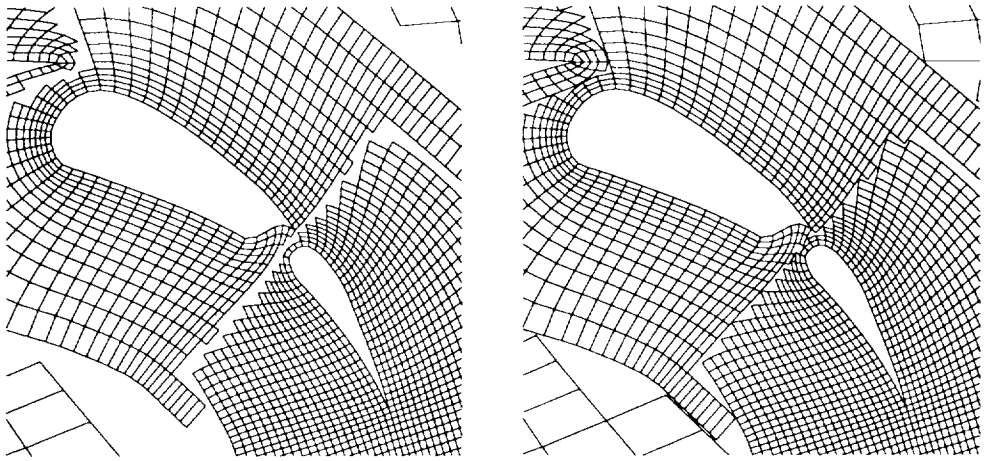


Figure 8: Structured cells advanced with (l) and without (r) proximity tests.

Front modification and reconnection.- If the *pface-cell1* combination passes both violation checks, the front is next modified to cover the cell. First, the cell to be covered, *cell1*, is flagged to indicate that it now lies behind the front. Next, faces in *cell1* not already on the front are added, and faces in *cell1* already on the front are removed. Finally, the new faces are connected to one another and to the neighbors of faces removed from the fronts. This task is straightforward for simple cases, but complicates quickly for odd combinations of converging fronts. For that reason, it was necessary to develop the general algorithm for front reconnection outlined in Figure 9.

edge →	n1	n2	n3	n4
Get a list of all faces and Cell1's (cells outside of front) on front using the edge.	f1,c1 f2,c2	f2,c2 f3,c3	f6,c4 f7,c3	f4,c1 f5,c4
Remove faces in Cell from list.	f1,c1	f3,c3	f6,c4 f7,c3	f4,c1 f5,c4
Add non-front faces of Cell using the edge to the list.	f8,c1	f10,c3	f10,c3 f9,c4	f8,c1 f9,c4
Pair sets of faces with the same Cell1. Remaining pair (if any) is also paired. For each pair, find corresponding edges. Link Face_a,Edge_a to Face_b,Edge_b.	f8-f1	f3-f10	f6-f9 f7-f10	f4-f8 f5-f9

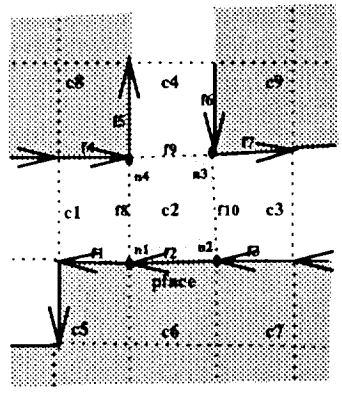


Figure 9: Front reconnection algorithm.

Front faces are reconnected via a four step procedure applied to each edge of *cell1* (*c*2 in Figure 9), the cell outside of *pface* to be covered by the front. In 2D (Figure 9), this amounts to a reconnection at the four nodes of the cell. In the first step a list of all faces and their corresponding *cell1*'s is formed from all front faces that use the edge. Next, any face/cell combinations with cells equal to the original *cell1* (*c*2) are

removed from the list. Third, any of the faces on *cell1* that do not lie on the front are added to the list. There will now be an even number of list entries, and these entries are grouped together on the basis of like cell numbers. On occasion when there remains two unmatched entries in the list with different cells, they are matched to one another. These paired cell/face combinations now refer to the connections needed to maintain a proper front linkage. For example, in the first column (edge $n1$) in Figure 9, faces $f8$ and $f1$ must be set to neighbor each other along their common edge, $n1$.

In the figure above, connections at edges $n3$ and $n4$ are broken in the upper left and upper right fronts and are reconnected with the faces in *cell1*. This is contrary to intuition, which would have the upper left and upper right front connections remain intact, and the faces on *cell1* connect to one another. Repeated front reconnections like this would result in fronts that collapse on themselves like a deflated balloon. In contrast, the algorithm outlined above will produce collapse-free fronts that will continually divide and reduce to minimal surface area. The difference is crucial when applying the unstructured advancing front algorithm described shortly.

Unused Component Removal

The structured portion of the advancing front algorithm finishes when each face on the front either lies on the outer edges of the multiple block grid or may advance no further due to close proximity to other regions of the front.

All cells, faces and nodes lying outside of the front at this point may be discarded, since they will not comprise part of the hybrid grid. Unused cells are determined by the absence of the flag set in cells overtaken by the front. After unused cells are removed, all faces and nodes in the remaining cells are flagged, and then all unflagged faces and nodes are removed.

Diagonal Swapping on the Final Structured Front

Before the regions encapsulated by the final structured front(s) will be filled with unstructured cells, it is necessary to massage the front to reduce the burden on the unstructured solver. In 3D, a side of a structured cell lying on the front will always be represented by a pair of complementary triangular faces, originally formed purely on the basis of their computational index. Since the unstructured solver has a natural bias to triangles of smaller maximum angle, the diagonal along each face pair on the front is compared to the imaginary diagonal along the two non-common nodes on the two faces. If the latter diagonal is shorter than the former, the quad region is retriangulated with diagonals swapped, so that the maximum angle on both triangles is reduced, as shown in Figure 10.

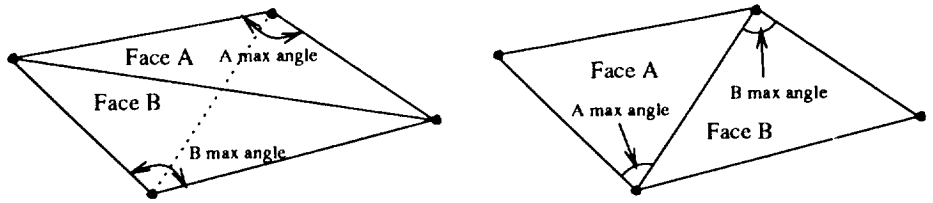
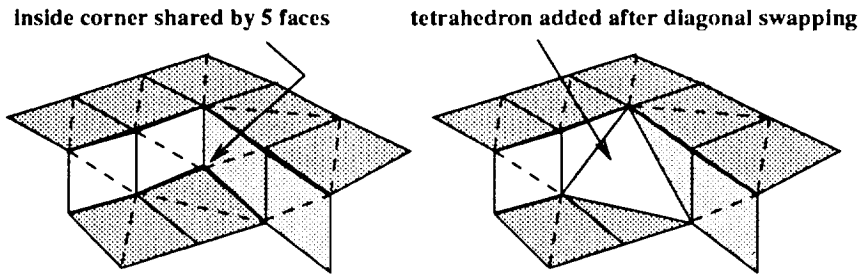


Figure 10: Swapping diagonals of complementary faces.

It is also helpful to swap diagonals on regions of the final front that represent inside corners of cells outside the front. Specifically, whenever three (or more) adjacent non-complementary faces on the front have the same *cell1* and share a common node, an inside corner situation exists, as depicted in Figure 11a. In these cases, the diagonals of the faces are swapped as necessary so that they do not touch the inside

corner. This creates a natural crevice that will later be filled with a tetrahedron by the unstructured solver (Figure 11b), thereby smoothing a formerly sharp corner of the front.



a. inside corner of an unused cell b. diagonals are swapped

Figure 11: Swapping diagonals on inside corners.

Refinement/Derefinement of Final Structured Front

The use of overlapping stretched structured grids to produce hybrid grids can produce starting fronts for the unstructured grid generator with significant variations in face sizes. When adjacent faces of a highly stretched structured grid cell are present on the front, a small face and a much larger face will be present. In addition a fine mesh around one component in the grid system may extend into the coarse mesh region of another component. Disparate sizes of faces in close proximity can lead to the generation of undesirable highly stretched or skewed triangles and tetrahedra. To lessen the face size disparity, local refinement and derefinement of the final structured front is added as a tool for further preprocessing the front for unstructured front advancement.

The derefinement procedure will merge faces on the front and their corresponding cells whenever the sizes of faces on the front are too small in comparison with a target size and whenever the aspect ratio of the structured grid cell behind the front is larger than a user specified value. Once the derefinement of the front is complete, the derefinement is propagated into the interior of the mesh my allowing a side of a cell to have at most two neighboring cells.

The refinement procedure will split faces on the front if the sizes of faces on the front are too large in comparison with a target size or if the size of a neighboring face. The procedure to split the cells behind the front has not yet been implemented.

Advancing Front With Unstructured Cells

When the the front has advanced as far as it can along structured cells, and after the final front has been massaged as described in the previous two sections, discretization of the domain is completed by advancing the front again, this time until it completely vanishes.

After the first advancing front application, there remains no more usable structured grid data beyond the front, and so in the second application the front is advanced from the more traditional, unstructured sense. Unfortunately, there are a whole new set of concerns to address during the unstructured application, since the front will now advance on the basis of geometric, rather than topological reasoning.

State variable.- Recall that a state indicator is stored at each face on the front, and its value reflects how

close the face and one or more of its neighboring faces are to forming a complete cell. For unstructured grid advancement, a minimal state value means that none of the neighboring faces are suitable for forming a cell with the face. The next state level means that a single neighboring face is suitable for forming a cell. This occurs when the angle between the two faces is less than a user-specified threshold, typically 135 degrees. The next highest state level occurs when two faces touching the face share a node and the angle between the build face and the two neighbors is less than the threshold. The highest state level occurs when the face and all its neighbors form a complete cell. Different front face states for a triangle and a quadrilateral are shown in Figure 12. States for the 3D cell forms, the tetrahedron and hexahedron, are defined in an analogous fashion.

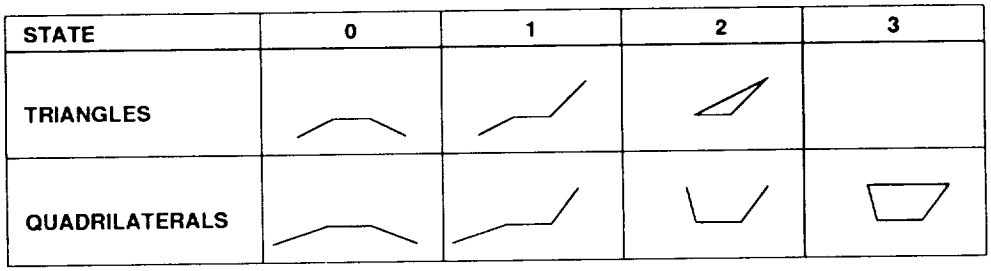


Figure 12: State levels for unstructured cells.

Target Face Size.- The present study uses a set of clustering points to specify a desired, or target, size for a cell face. Each clustering point is given by a spatial location, the desired size of a face, d_i at that location (x_i, y_i, z_i) and a clustering parameter, C_i, that controls the spatial variation away from the point. The target size, t, at an arbitrary point in space (x, y, z) is given by

$$t = \min_i d_i + C_i R_i^2$$

where $R_i^2 = (x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2$ and $\min_i$ represents the minimum value for all the clustering points.

The clustering points can be specified by the user or set to the size and locations of the faces on a set of boundary surfaces. The latter method ensures a smooth variation in faces away from each boundary with a minimum of user inputs.

The target size can also be specified by interpolating field values from a cartesian background grid similar to the approach of [13]. This would be the preferred approach for 3D cases due to the large number of faces on the surfaces of the geometry.

Candidate face selection.- Just as it was for structured front advancement, the unstructured grid generated by the advancing front method is significantly influenced by the criteria used for *pface* selection. During the unstructured application, the advancing face *pface* is set equal to the first face returned from the ordered set of criteria below:

1. the first face that is at the maximum state (forming a closed cell with neighboring front faces) or the maximum state minus one (lacking only one face to form a closed cell).
2. the face that forms the minimum angle (below the threshold angle) with a neighboring front face.
3. the smallest face on the front.

A degree of uniformity and smoothness may be incorporated into an unstructured mesh by assigning a cycle number to each front face and by advancing a face only if it has a lower cycle number than currently being used. As described earlier, when no suitable faces are found, the cycle number is incremented. This cycling strategy seems to work best when filling large areas/volumes that have smooth boundaries. Hence, it is probably not appropriate for use in this application- namely, filling in the jagged regions between overlapping structured grids.

Candidate Cell selection.- The most critical step in unstructured front advancement is the determination of the validity of the candidate cell, *cell1*. Candidate cells will still be deemed invalid if their faces intersect any others on the front, but they will also be rejected if they lie too close to others on the front. Rejection is necessary to filter out highly stretched or skewed cells from the hybrid grid.

The four step procedure below is currently used to arrive at a suitable cell emanating from *pface*, the building face.

1. If *pface* is at the maximum state, form the cell from its neighboring faces.
2. Generate a node perpendicular to *pface* using the target cell size for the face. If the node is not too close to another face, if the node is not within the bounding sphere of a face on the front, and if the faces formed by the node do not intersect the front, form the cell from *pface* and the generated node. In 2d, try to form an even better cell using Delaunay criteria as suggested by Merriam [14]. Use the node that is closest to the circumsphere center of the candidate cell to form a new candidate cell.
3. Use the nodes on neighboring faces that raised *pface* to its state to form a cell.
4. Form the cell by using nearby nodes on the front within some radius of *pface*.

Note that no attempt is made to maintain a Delaunay [15] grid. Though the Delaunay criteria will produce a grid with triangles that are the most equilateral, it is overly restrictive in 3D as it produces flat or planar cells in regions of coplanar nodes. Since the structured grids forming the initial unstructured front will typically have numerous regions of planar nodes, the Delaunay method would connect these nodes to form flat cells.

The nearly planar nodes and faces forming the initial unstructured front are especially troublesome when the faces surround a point that was a corner of a structured cell. See for example Figure 13. If the wrong choice for a candidate cell is made and accepted, the cell may well contain a face that is nearly planar with other faces on the front. A different choice for the starting face and hence the candidate cell would have eliminated the possibility of the cell with the planar face from even being considered. The present method begins by finding all the nodes with 6 faces surrounding the node. If two neighboring faces in the list for a node are found to be planar, then a cell is constructed from one of the faces. The same procedure is then applied to nodes with 5 and 4 surrounding faces.

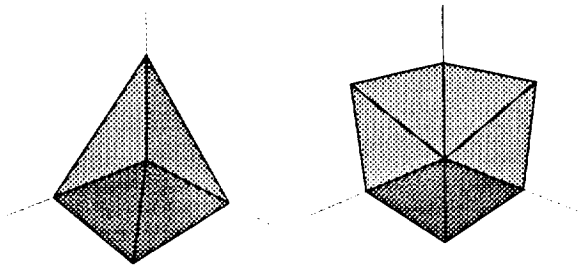


Figure 13: Node Surrounded by 4 faces(l) and 6 faces(r)

EXAMPLES

The present approach is first illustrated using a two-dimensional multi-element airfoil as shown in Figure 14. Figure 15 presents the overlapping structured system, which consists of an "O-type" structured mesh around the main airfoil section and "C-Type" meshes around the leading and trailing edge flaps. The structured mesh with the overlapping cells removed is shown in Figure 16. Close up views of the leading and trailing edge regions are shown in Figure 17 and clearly illustrate the difficulties in using the front left by removing cells from a stretched and clustered mesh. The cells in the leading edge region of the main airfoil grid are stretched and have a high aspect ratio. The front will have a small face adjacent to a larger face, corresponding to the sides of one of the stretched cells. In the trailing edge region the fine mesh around the last trailing edge flap has cells, and hence faces on the front, that are in the coarse region of the main airfoil grid. Clearly it is advantageous to fill the void with a smoothly varying unstructured mesh.

Two different approaches have been tested in the present effort and are illustrated in Figures 18 and 19. The first approach uses refinement of the unstructured grid starting front to ensure a smooth variation in the faces on the front. Figure 18 shows that some of the sides of the quadrilaterals adjacent to the unstructured cells have been split into smaller faces without splitting the cell. The companion flow solver was developed to allow an arbitrary number of faces per cell. Refining the front may add too many cells so a second approach of derefinement has been tested. Figure 19 show the results for the case where the structured grid front and the cells behind the front have been derefined to provide a smoother distribution of cell sizes. Figures 20 and 21 show the final complete hybrid grids for the multi-element airfoil when refinement and derefinement strategies are used. The derefinement procedure has significantly reduced the number of cells in comparison to the refinement case. The preferred approach will obviously depend upon the particular flow conditions in the region of refinement or derefinement.

The results for a three-dimensional case are illustrated for a aircraft store in close proximity to a pylon. For simplicity the wing geometry is not considered in this example. The pylon and store geometries are shown in Figure 22 and a portion of the overlapping structured grids around the geometries is shown in Figure 23. The portion of the front resulting from removal of the overlapping portion of the pylon mesh is shown in Figure 24. and for the removal of the overlapping portion of the store mesh is shown in Figure 25. In this particular case the unstructured grid starting front will include a portion of the store surface since the store and pylon are in such close proximity. The tetrahedral unstructured mesh filling the void between the store and pylon meshes is not presented here due to the difficulty in viewing three-dimensional unstructured grids.

CONCLUSIONS

This paper has introduced a new 3-dimensional hybrid grid generation technique based on advancing front ideas applied to both structured and unstructured grids. The advantages of the scheme have been shown to be that structured grids may be generated around individual grid components independently, with any overlap or gaps between grids removed and/or filled automatically. This dramatically reduces grid generation time, but still provides a locally structured grid near the geometry surfaces. Further, the structured character of cells near the body markedly reduces the number of cells needed to resolve the flowfield.

Future effort is aimed at improving the efficiency of the hybrid generation time and also the quality of the grid. Efficiency issues include the improvement of the quadtree/octree searching routines, and grid quality issues include modifications to the face selection criteria, refinement of the front proximity definition for converging fronts, and extension of the scheme to allow non-enclosed initial fronts, such as in problems with a plane of symmetry.

ACKNOWLEDGEMENTS

This work was sponsored by the United States Air Force Wright Laboratory, Armament Directorate, Eglin AFB, FL under contract F08630-93-C-0074.

REFERENCES

1. Belk, D.M., and Maple, R.C., "A New Approach to Domain Decomposition: The Beggar Code," Proceedings of the 2nd Overset Composite Grid and Solution Technology Symposium, October, 1994.

2. Weatherill, N.P., "On the Combination of Structured-Unstructured Meshes," **Numerical Grid Generation in Computational Fluid Mechanics '88**, Sengupta et al., eds., Pineridge Press, 1988.

3. Shaw, J.A., Georgala, J.M., May, N.E., and Pocock, M.F., "Appliction of Three-Dimensional Hybrid Structured/Unstructured Grids to Land, Sea and Air Vehicles," **Numerical Grid Generation in Computational Fluid Dynamics and Related Fields**, N.P. Weatherill, et al., ed., pp. 687-698, Pineridge Press, 1994.

4. Marcum, D.L., "Generation of Unstructured Grids for Viscous Flow Applications," AIAA Paper 95-0212, January, 1995.

5. Kallinderas, Y., Khawaja, A. and McMorris, H., "Hybrid Prismatic/Tetrahedral Grid Generation for Complex Geometries," AIAA Paper 95-0211, January, 1995.

6. Kao. K.-H., and Liou, M.-S., "Direct Replacement of Arbitrary Grid-Overlapping by Nonstructured Grid," AIAA Paper 95-0346, January, 1995.

7. Wey, T.C., "Development of a Mesh Interface Generator for Overlapped Structured Grids," AIAA Paper 94-1924, 1994.

8. Samet, H.: "The Quadtree and Related Hierarchical Data Structures", **Computing Surveys**, 16(2), pp. 187-260, 1984.

9. Chawner, J.R., and Steinbrenner, J.P., "Automatic Structured Grid Generation Using GRIDGEN (Some Restrictions Apply)", Proceedings of the 1995 Workshop on Surface Modeling, Grid Generation, and Related Issues in CFD Solutions, NASA CP, 1995.

10. Steinbrenner, J.P., and Chawner, J.R., **The GRIDGEN Version 8 Multiple Block Grid Generation Software**, MDA Engineering Report 92-01, 1992.

11. Foley, J., van Dam, A., Feiner, S., and Hughes, J., **Computer Graphics: Principles and Practice, Second Edition**, Addison-Wesley, 1990.

12. Bishop, D.G.; and Noack, R.W.: "An Implicit Flow Solver With Upwind Differencing For Three-Dimensional Hybrid Grids, " AIAA Paper 95-1707CP, 12th AIAA Computational Fluid Dynamics Conference, 1995.

13. Pirzadeh, S., "Structured Background Grids for Generation of Unstructured Grids by Advancing-Front Method," **AIAA J.**, 31(2), pp. 257-265, February, 1993.

14. Merriam, M.L., "An Efficient Advancing Front Algorithm for Delaunay Triangulations," AIAA Paper 91-0792, 1991.

15. Bowyer, A., "Computing Dirichlet Tesselations," The Computer J., Vol. 24, No. 2, pp 162-166, 1981.

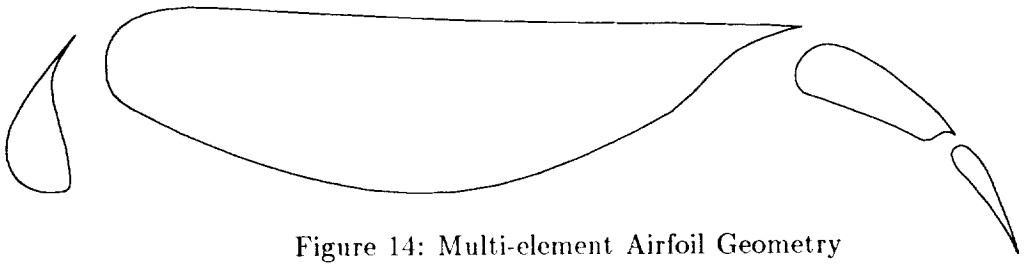


Figure 14: Multi-element Airfoil Geometry

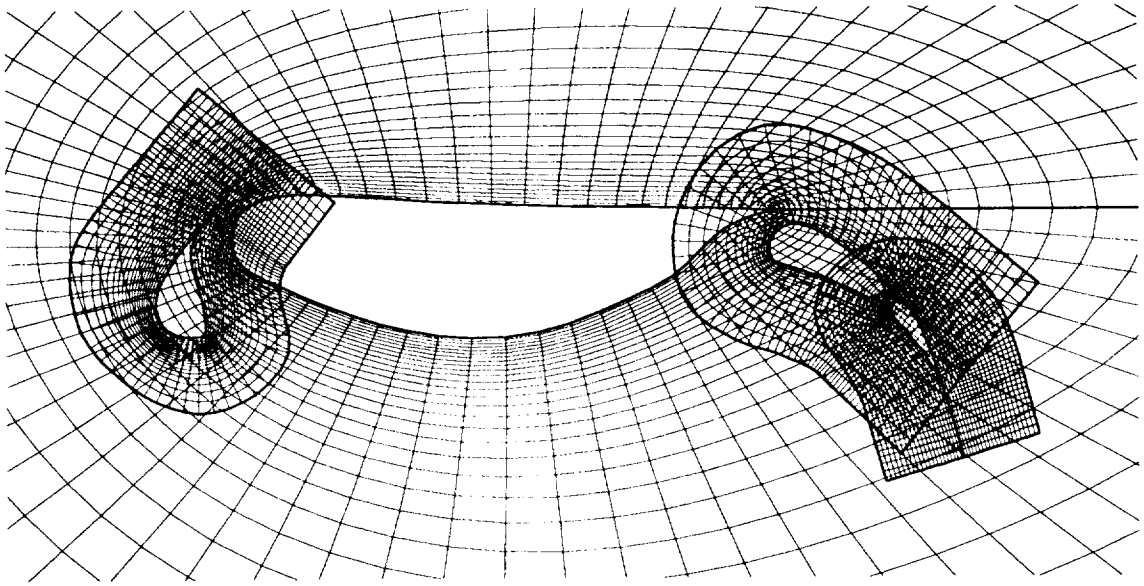


Figure 15: Set of Overlapping Structured Grids for Multi-element Airfoil Geometry

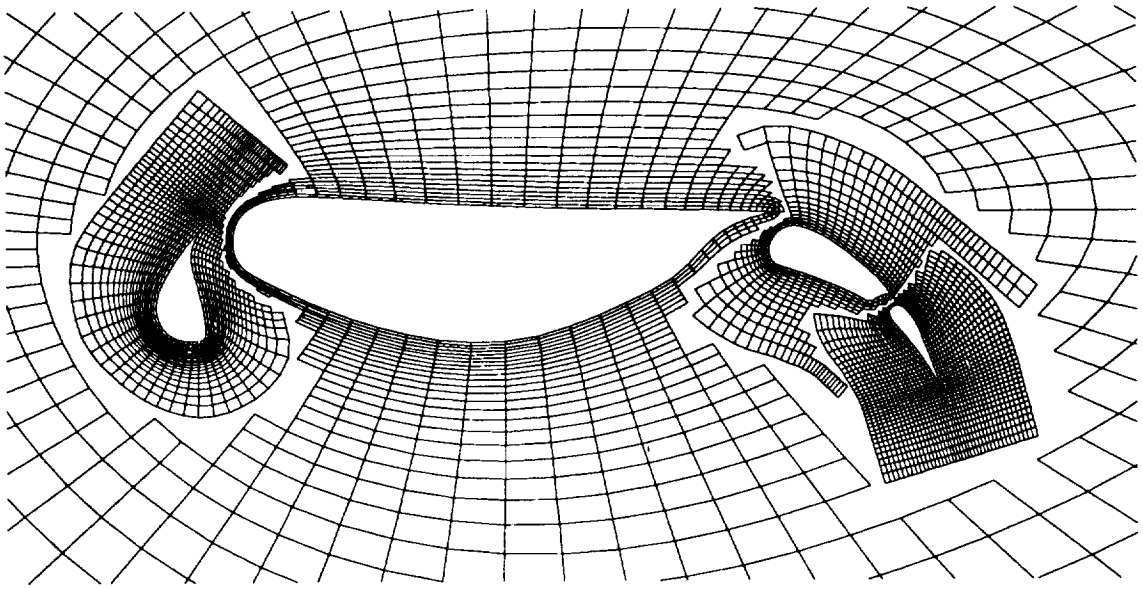


Figure 16: Structured Grids with Overlap removed for Multi-element Airfoil Geometry

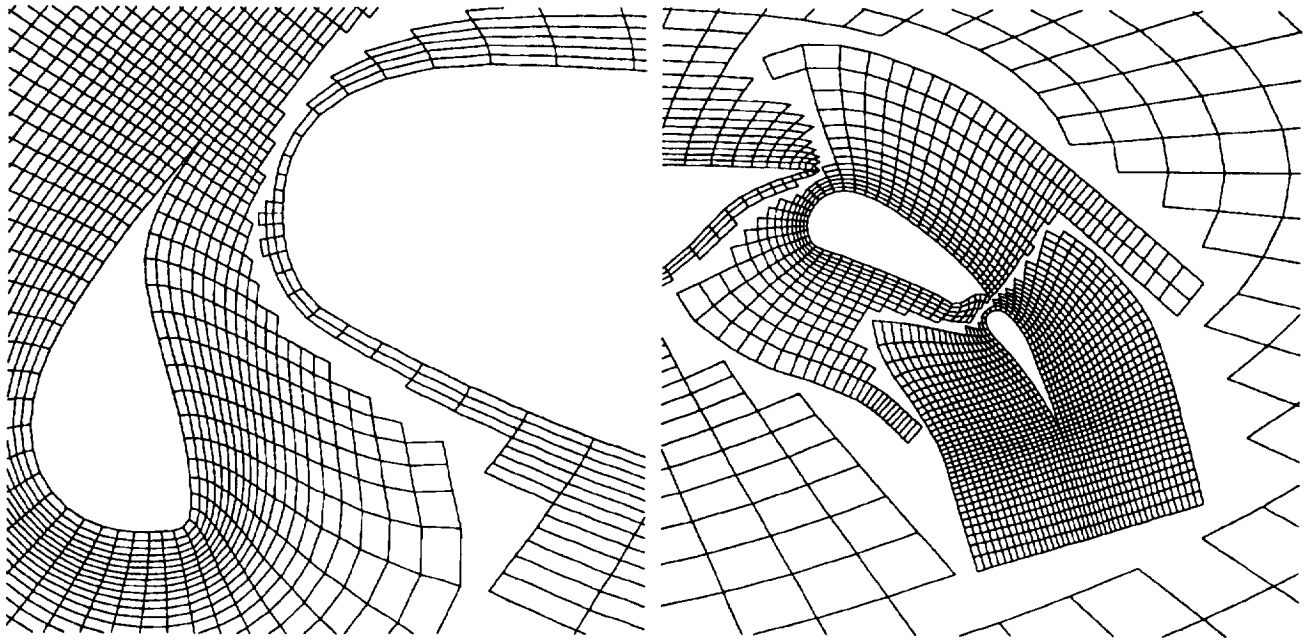


Figure 17: Closeup of Leading(l) and Trailing(r) Edge Flap Region for Structured Grids with Overlap removed

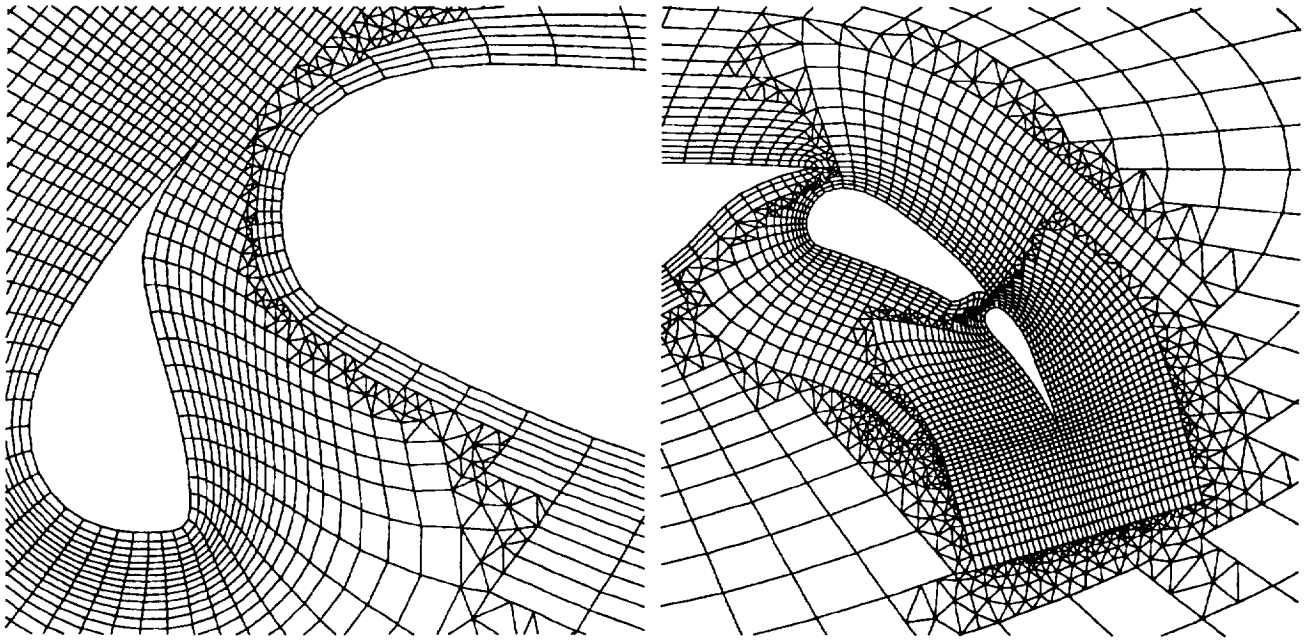


Figure 18: Closeup of Leading(l) and Trailing(r) Edge Flap Region of Hybrid Grid With Refinement of Front

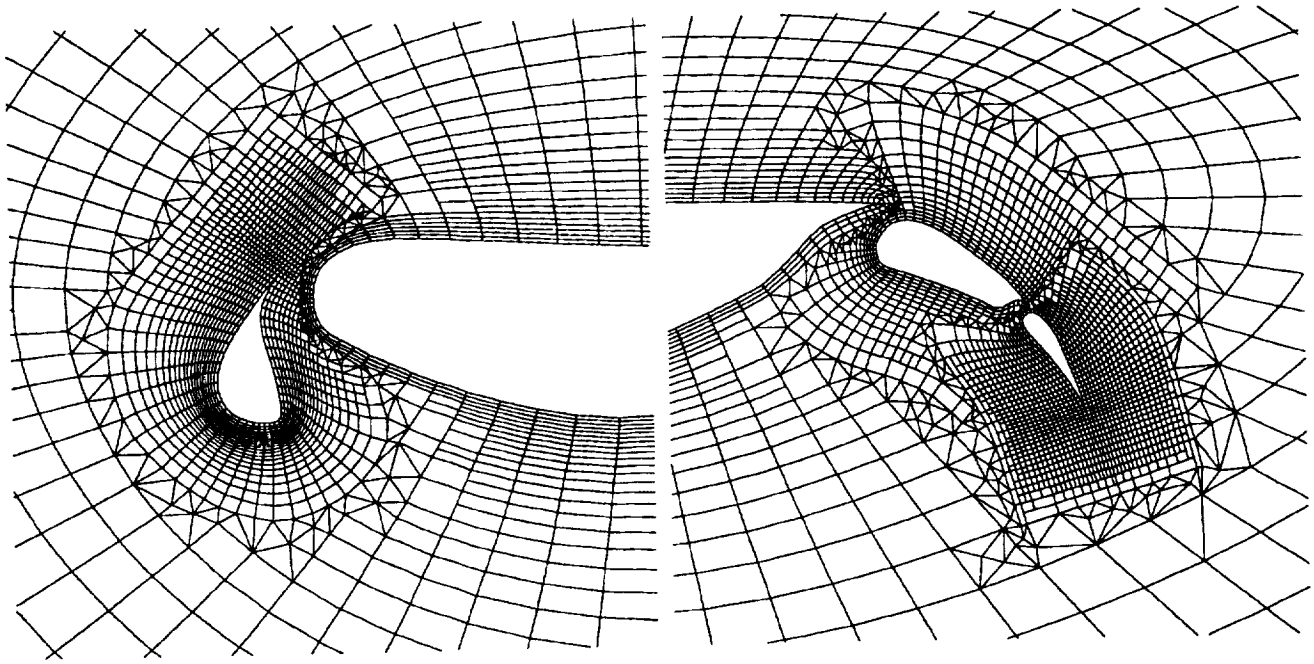


Figure 19: Closeup of Leading(l) and Trailing(r) Edge Flap Region of Hybrid Grid With Derefinement of Front

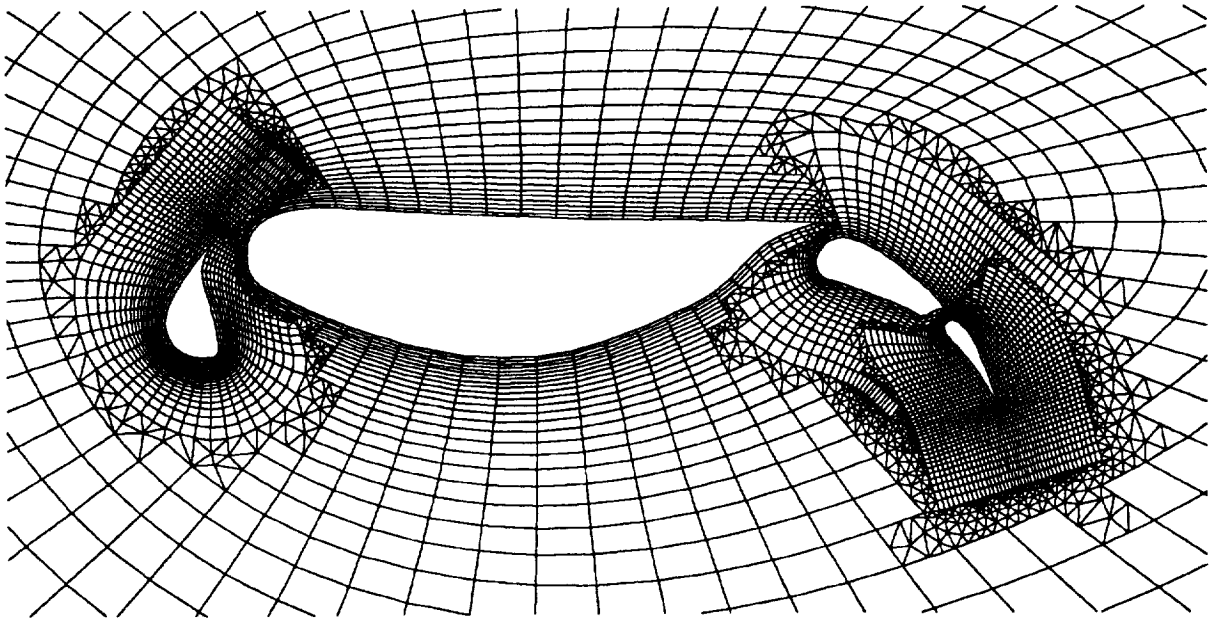


Figure 20: Final Hybrid Grid for Multi-Element Airfoil with Refinement of Front Between Structured and Unstructured Grid Regions

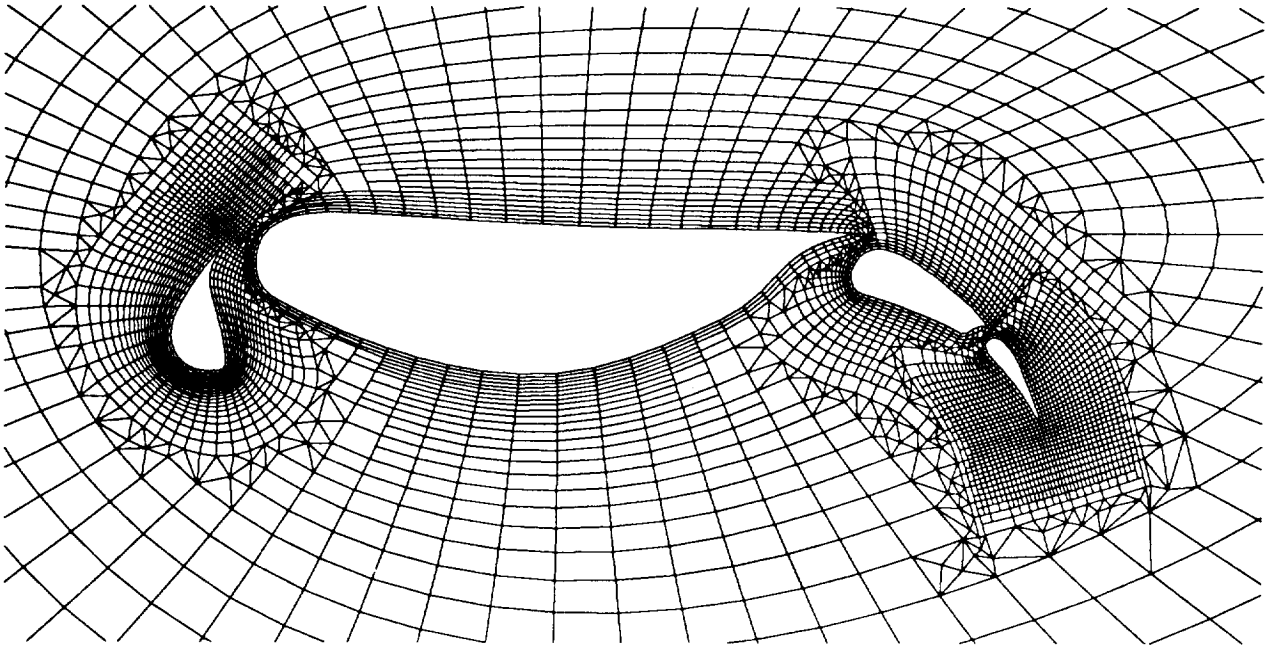


Figure 21: Final Hybrid Grid for Multi-Element Airfoil with Derefinement of Front Between Structured and Unstructured Grid Regions

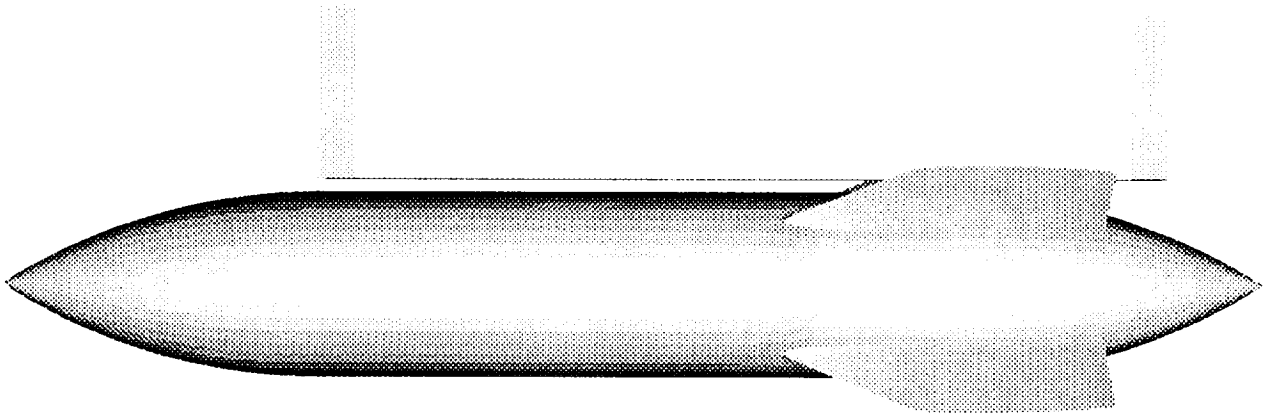


Figure 22: Pylon and Store Geometry

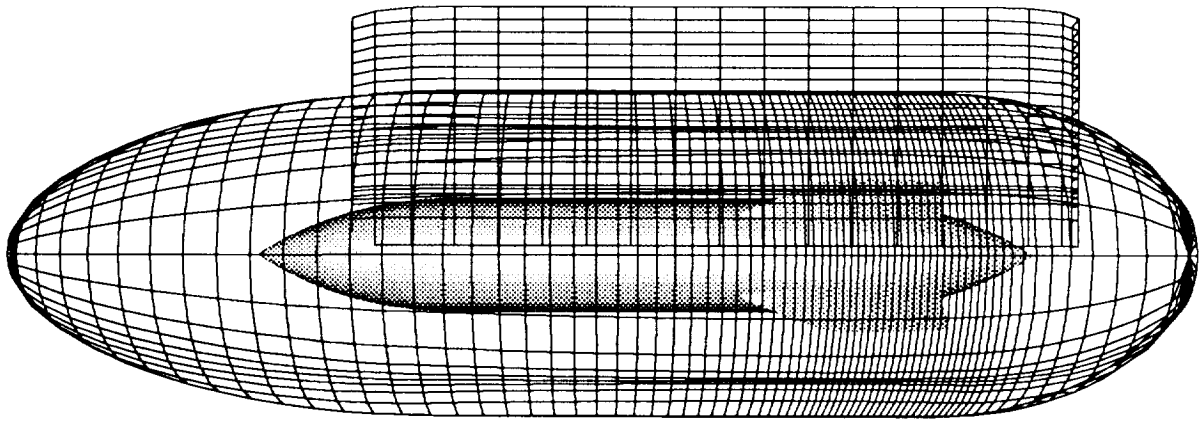


Figure 23: Pylon and Store with Portion of Grids Surrounding the Pylon and Store

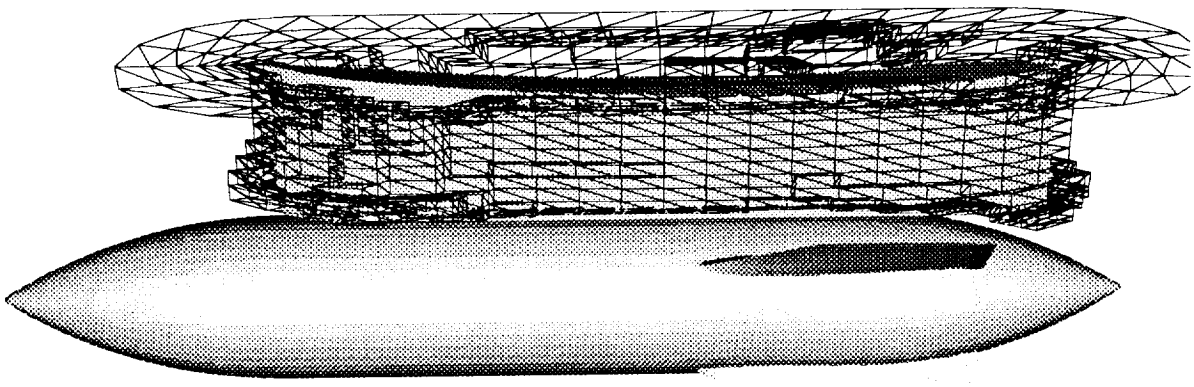


Figure 24: Store with Front from Removal of Overlapping Portion of the Pylon Mesh

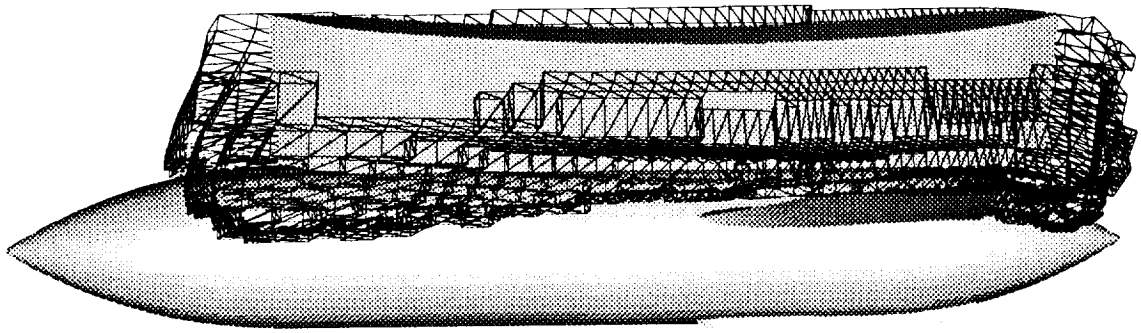


Figure 25: Store and Pylon with Front from Removal of Overlapping Portion of the Store Mesh

TECHNOLOGY ASSESSMENT